Lecture-4

Introduction to Unix:
More Commands, Boot-up Actions and X Window

What You Will Learn

- We continue to give more information about the fundamental commands of the Unix operating system.
- We also give some introductory information about the the system's bootup sequences.
- As before, everything told for Unix here is applicable to the Linux operating system also.

Working with Unix

- UNIX is a powerfull system for those who know how to harness its power. In this chapter, we'll try to describe various ways to use Unix's *shell*, *bash*, more efficently.

Wildcards

By now you are doubtless tired of typing every letter of each filename into your system for each example. There is a better and easier way! Just as the special card in poker can have any value, UNIX has special characters that the various shells (the command line interpreter programs) all interpret as *wildcards*. This allows for much easier typing of patterns.

* acts as a match for any number and sequence of characters,

? acts as a match for any single character.

- a lone * acts as a match for all files in the current directory (in other words, ls * is identical to ls),

- a single ? acts as a match for all one-character-long filenames in a directory (for instance, ls ?, which will list only those filenames that are one character long).

You might want to copy all the files beginning with data into a directory called backup. You could do this by either running many cp commands, or you could list every file on one command line. Both of these methods would take a long time.

- A better way of doing that task is to type:

```
/home/larry/report# ls -F
1993-1   1994-1     data 1    data 5
1993-2   data-new   data 2
/home/larry/report# mkdir ~/backup
/home/larry/report# cp data* ~/backup
/home/larryreport# ls -F ~/backup
data-new   data 1   data 2    data 5
/home/larry/report#
```

- The asterix (*) told cp to take all of the files beginning with data and copy them to ~/backup.

What Really Happens ?

- There are a couple of special characters intercepted by the shell, bash.
- The character "*", an asterix, says "replace this word with all the files that will fit this specification".
- So, the command cp data* ~/backup, like the one above, gets changed to

  cp data-new data 1 data 2 data 5 ~/backup

  before it gets run.

• echo

• echo is an extremely simple command; it echoes back, or prints out, any parameters.

• The echo command simply writes back to the screen anything specified.

```
/home/larry# echo How are you?
How are you?
/home/larry#
```

```
/home/larry# cd report
/home/larry/report# ls –F
1993-1   1994-1    data1   data5
1993-2   data-new  data2
/home/larry/report# echo 199*
1993-1   1993-2   1994-1
```

- The Question Mark

  - In addition to the asterix, the shell also interprets a question mark as a special character.

  - A question mark will match one, and only one character. For instance, ls /etc/?? will display all two letter files in the /etc directory.

- Time Saving with bash
  - Command-Line Editing

- command line editing allows you to edit your previously executed commands by using arrow keys.
- you can use the the arrow keys to move back, type the correct information.
- There are many special keys to help you edit your command line, For instance, Ctrl t character flips two adjacent characters.

• Command and File Completion

- Another feature of bash is automatic completion of your command lines.
- command and file completion lets you type long files without typing every word of it

```
[bagriyanik@hyperion bagriyanik]$ > ls -F
this_is_a_long_file i
```

• type [yildirimd1]> cp th then press and release TAB key rest of the filename shows up on the command line

• when you press TAB, bash looks at what you've typed and looks for a file that starts like it

• if you try a completion and bash beeps, there are more than one files starting with th . (in this case type more letters)

The Standard Input and The Standard Output

Unix Concepts

- The unix operation system makes it very easy for programs to use the terminal.
- When a program writes something to your screen, it is using something called *standard output*. Standard output, abbreviated as *stdout*, is how the program writes things to a user.
- The name for what you tell a program is *standard input* (*stdin*).
- It's possible for a program to communicate with the user without using standard input or output, but most of the commands use stdin and stdout.

- most commands write their output to stdout and error messages to stderr.
- the file descriptors stdin, stdout, and stderr are the three files that are automatically opened when you log in.
- normally both stdout and stderr are the terminal screen by default.
- In this section, we're going to examine three ways of fiddling with the standard input and output:
    - input redirection,
    - output redirection,
    - pipes.

• Output Redirection

- A very important feature of Unix is the ability to redirect output.
- This allows you, instead of viewing the results of a command, to save it in a file or send it directly to a printer. For instance, to redirect the output of the command ls /usr/bin, we place a > sign at the end of the line, and say what file we want the output to be put in:

```
/home/larry# ls
/home/larry# ls -F /usr/bin > listing
/home/larry# ls
listing
/home/larry#
```

- As you can see, instead of writing the names of all the files, the command created a totally new file in your home directory.

- it is possible to redirect stdout to a file using the > operator
- redirection allows the user to append to the end of an existing file using >>

Input Redirection

- Like redirecting standard output, it is also possible to redirect standard input.
- Instead of a program reading from your keyboard, it will read from a file.
- Since input redirection is related to output redirection, it seems natural to make the special character for input redirection be <.
- It too, is used after the command you wish to run. This is generally useful if you have a data file and a command that expects input from standard input.

• normally stdin is the input to a command, and it is read

from the terminal.

• sometimes, you might want the contents of a file to be stdin; e.g., mailing the contents of a file to someone

• stdin may be redirected so that it comes from a file using the < operator:

[hyperion]$ mail bagriyani3@hyperion.itu.edu.tr < message.txt

• the contents of the file message.txt becomes the stdin for the mail program, instead of the user actually typing the body of the mail message.

- input and output descriptors

file descriptors			
file descriptor	descriptor name	file descriptor number	normal input or output location
stdin	standard input	0	keyboard
stdout	standard output	1	terminal
stderr	standard error	2	terminal

- redirection operators

operator	action
<	open the following file as *stdin*
>	open the following file as *stdout*
>>	append to the following file
>2	open the following file as *sterr*

- The Pipe

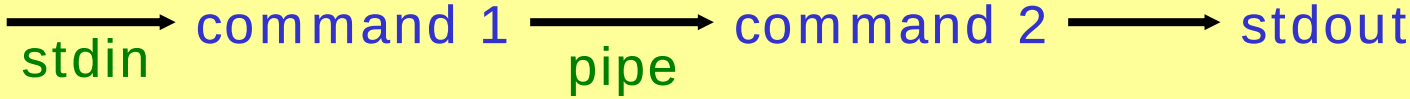
  - Many Unix commands produce a large amount of information. For instance, it is not uncommon for a command like

    ls /usr/bin

    to produce more output than you can see on your screen.
  - In order for you to be able to see all of the information that a command like ls /usr/bin, it's necessary to use another Unix command, called *more*.
  - more is named because that's the prompt it originally displayed: - -more- -.

- Unix supplies a much cleaner way of doing that.
- You can just use the command

  ls /usr/bin | more
- the above command lets you view the listing on a per page basis
- The character " | " indicates a **pipe**. Like a water pipe, a Unix pipe controls flow. Instead of water, we're controlling the flow of information!



- A useful tool with pipes programs called **filters**.
- A filter is a program that reads the standard input, changes it in some way, and outputs to standard output.
- More is a filter.

  more reads the data that it gets from standard input and display it to standard output one screen at a time, letting you read the file. more isn't a great filter because its output isn't suitable for sending to another program

some filters are:

- cat
- sort
- head
- tail

- pipes can connect a series of commands together.

- pipe and filters
  - suppose file p1.txt has the following contents

```
[hyperion]$ more p1.txt
497381 yusuf bagdu 70
498420 zeynep erenay 100
497404 okan kurtkaya 100
497420 ilknur colak 70
497402 atilla kilic 100
[hyperion]$ grep 100 p1.txt
498420 zeynep erenay 100
497404 okan kurtkaya 100
497402 atilla kilic 100
```

- [hyperion]$ grep 100 p1.txt | sort : list lines that contains 100 in p1.txt and sort the result with the first column
- [hyperion]$ grep 100 p1.txt | sort -k 2 -r : list lines that contains 100 in p1.txt and sort the result with the 2nd column in reverse order
- [hyperion]$ grep 100 p1.txt | sort -k 2 -r > p11.txt

- Multitasking
  - Using Job Control

- Job control refers to the ability to put processes in the background and bring them to the foreground again.
- That is to say, you want to be able to make something run while you go and do other things, but have it be there again when you want to tell it something or stop it.
- In Unix, the main tool for job control is the shell - it will keep track of jobs for you, if you learn how to speaks its language.

- The two most important words in that language are
  - fg, for foreground, and
  - bg, for background.
- To find out how they work, use the command *yes* at a prompt.
  /home/larry# yes
- You should see various messages about termination of jobs - nothing dies quietly, it seems.
- The following table gives a summary of commands and keys used in job control.

- **A summary of commands and keys used in job control**

1. fg % job : returns a job to the foreground
2. & : when it is added at the end of the command line, the command will run in the background automatically
3. bg % job : causes a suspended job to run in the background
4. kill % job or PID : causes a background job either suspended or running to terminate.
   PID (**p**rocess **ID** number) either the job number (preceded by %) or PID (no % is necessary)
5. Jobs : lists information about the jobs currently running or suspending
6. Ctrl c : generic interrupt character, if you type it, it will kill the program when it is running in the foreground
7. Ctrl z : causes a program to suspend

- job control with PIDs

- Each process has its own unique PID number

  ps : it will list all running processes including your shell

  ps aux : aux is an option here meaning:

    a: list processes belonging to any user not just yours

    u: will give additional information about the processes

    x: list proceses that don't have a terminal associated with them

Virtual Consoles: Being in Many Places at Once

- Linux supports **virtual consoles**.
- These are a way of making your single machine seem like multiple terminals, all connected to one Linux kernel.
- Using virtual consoles is one the simplest things about Linux: there are "hot keys" for switching among the consoles quickly.
- To try it, log in to your Linux system, hold down the left Alt key, and press F2.
- You should find yourself at another login prompt. Don't panic: you are now on virtual console (VC) number 2! Log in here and do some things - a few ls's or whatever - to confirm that this a real login shell. Now you can return to VC number 1, by holding down the left Alt and pressing F1. Or you can move on to a third VC, in the obvious way (Alt-F3).

• Boot-up Actions

- You may have previous experience with MS-DOS or other single user operating systems, such as OS/2 or the Macintosh.
- In these operating systems, you didn't have to identify yourself to the computer before using it; it was assumed that you were the only user of the system and could access everything.
- Unix is a multi-user operating system. To tell people apart, Unix needs a user to identify him or herself by a process called logging in.

- Power to the Computer

  - The first thing that happens when you turn an Intel is that the BIOS (Basic Input/Output System) executes.
  - If there isn't a floopy disk in the drive, the BIOS looks for a master boot record (MBR) on the hard disk. It will start executing the code found there, which loads the operating system. On Linux systems, LILO (the **LI**nux **LO**ader), can occupy the MBR position, and will load Linux.

Linux Takes Over

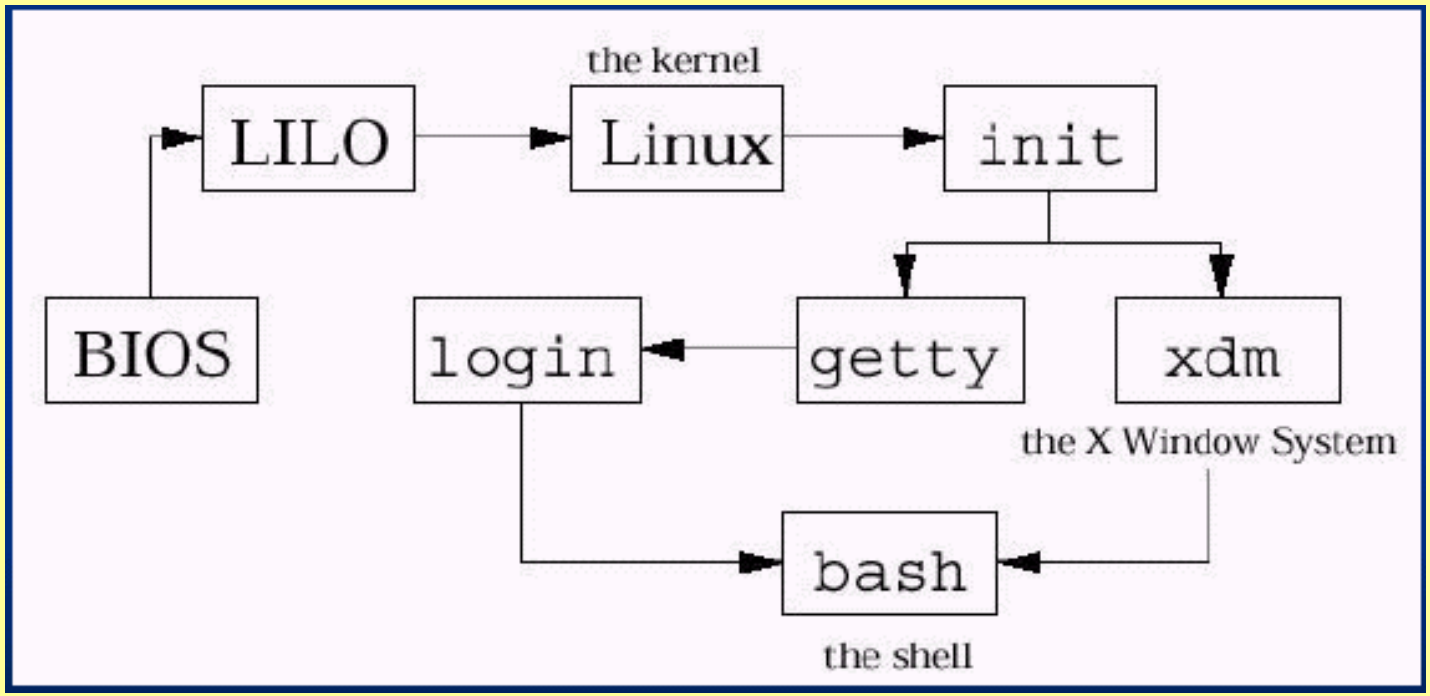
- After the BIOS passes control to LILO, LILO passes control to the Linux kernel.
- A kernel is the central program of the operating system, in control of all other programs. The first thing that Linux does once it starts executing is to change to protected mode.
- Linux looks at the type of hardware it's running on. It wants to know what type of hard disks you have.
- During boot-up, the Linux kernel will print variations on several messages. You can read about the messages in kernel-messages
- The kernel merely manages other programs, so once it is satisfied everything is okay, it must start another program to do anything useful. The program the kernel starts is called ***init***. After the kernel starts ***init***, it never starts another program. The kernel becomes a manager and a provide, not an active program

Some things that might happen once *init* is started:

1. The file systems (files on the hard disk) might be checked.
2. Special routing programs for networks are run. These programs tell your computer how it's suppose to contact other computers.
3. Temporary files left by some programs may be deleted.
4. The system clock can be correctly updated.

- After *init* is finished with its duties at boot-up, it goes on to its regularly scheduled activities.
- *init* can be called the parent of all process on a Unix system.
- A process is simply a running program. Since one program can be running two or more times, there can be two or more processes for any particular program.
- On your Linux system, what *init* runs are several instances of a program called *getty*.
- *getty* is the program that will allow a user to login and eventually calls a program called *login*.

- boot-up in Linux



The X Window System

- This chapter only applies to those using the X Window System. If you encounter a screen with multiply windows, colors, or a cursor that is only movable with your mouse, you are using X.

Starting and Stopping the X Window System

Starting X

- Even if X doesn't start automatically when you login, it is possible to start it from the regular text-mode shell prompt.

- There are two possible commands that will start X, either *startx* or *xinit*.

• Exiting X

- Depending on how X is configured, there are two possible ways you might have to exit X.
- The first is if your window manager controls whether or not X is running. If it does, you'll have to exit X using a menu. To display a menu, click a button on the background.
- The important menu entry sould be "Exit Window Manager" or "Exit X" or some entry containing the word "Exit".
- The other method would be for a special *xterm* to control X. If this is the case, there is probably a window labeled "login" or "system xterm". To exit from X, move the mouse cursor into that window and type "exit".
- If X was automatically started when you logged in, one of these methods should log you out.
- If you started X manually, these methods should return you to the next mode prompt. If you wish to *logout* at this prompt.

• What is the X Window System ?

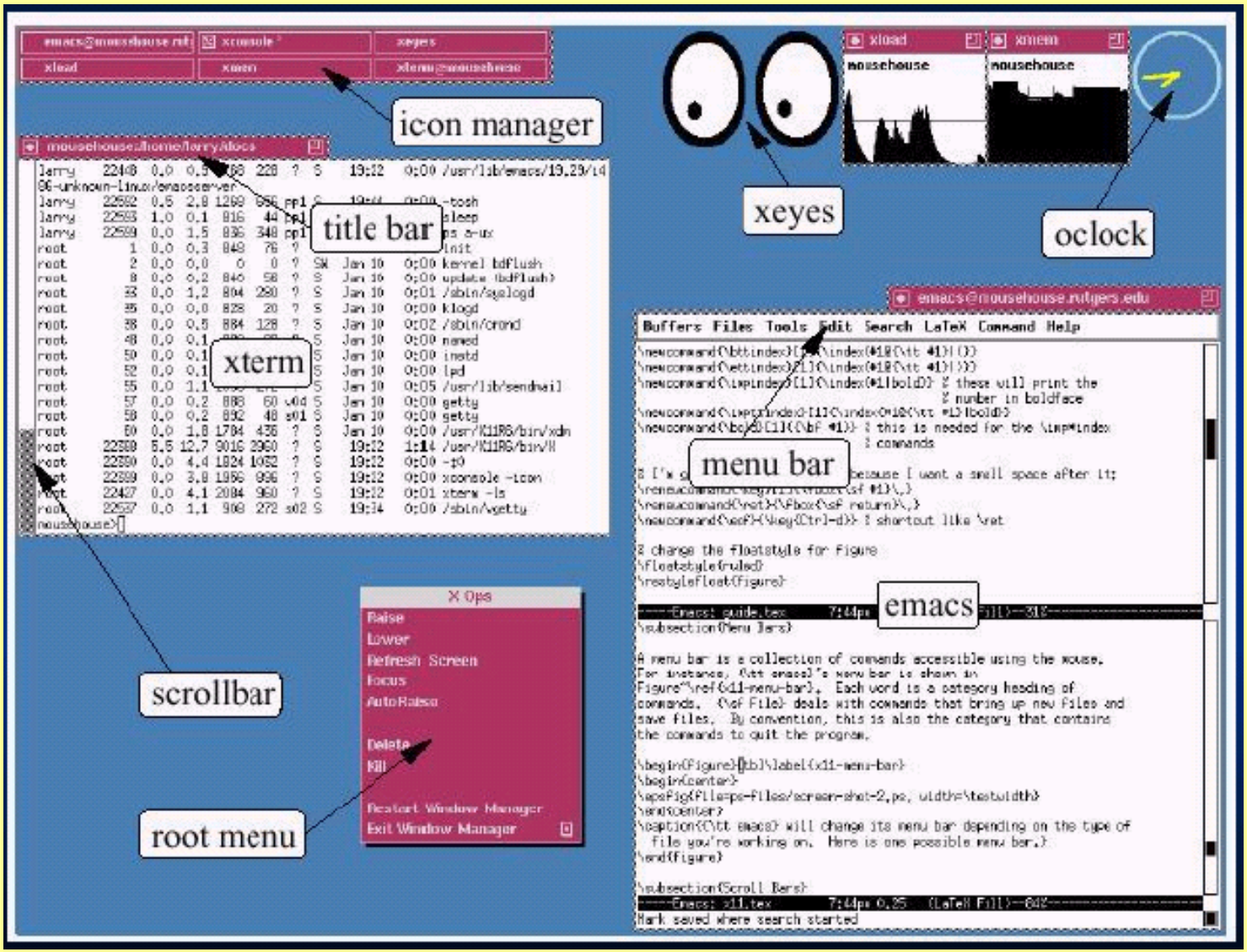
- The X Window System is a distributed, graphical method of working developed primarily at the Massachusetts Institute of Technology.
- There are two terms when dealing with X that you should be familiar.
  - The *client* is a X program. For instance, *xterm* is the client that displays your shell when you log on.
  - The server is a program that provides services to the client program. For instance, the server draws for *xterm* and communicates with the user.

- A third term you should be familiar with is the *window manager*. The window manager is a special client.
- The window manager tells the server where to position various windows and provides a way for the user to move these windows around.
- The server does nothing for the user. It is merely there to provide a buffer between the user and the client.

What's This on my Screen ?

- When you first start X, several programs are started. Then, several clients are usually started.
- It is likely that among these clients are a window manager, either *fvwm* or *twm*, a prompt, *xterm*, and a clock, *xclock*.
- **X Clock**
  xclock functions exactly as you'd expect it would. It ticks off the seconds, minutes and hours in a small window.
- **X Term**
  The window with a prompt in it (looks like /home/larry#) is being controlled by a program called *xterm*. Xterm is a complicated program.

X Window


icon manager
title bar
xterm
xeyes
oclock
menu bar
emacs
scrollbar
root menu
X Ops
Raise
Lower
Refresh Screen
Focus
AutoRaise
Delete
Kill
Restart Window Manager
Exit Window Manager

Window Managers

- On Linux, there are two different window managers that are commonly used.
- One of the them, called *twm* (Tab Window Manager).
- Other one is *fvwm* .
- Both twm and fvwm are highly configurable.
- *twm* is larger than *fvwm*.

When New Windows are Created

- There are three possible things a window manager will do when a new window is created.
- It is possible to configure a window manager so that an outline of the new window is shown, and you are allowed to position in on your screen. That is called *manual placement*.
- It is also possible that the window manager will place the new window somewhere on the screen by itself. This is known as *random placement*.
- Finally, an application will ask for a specific spot on the screen, or the window manager will be configured to display certain applications on the same place of the screen all the time.

• Focus

- The window manager controls some important things. The first thing you'll be interested in is *focus*.
- The focus of the server is which window will get what you type into the keyboard.
- In X the focus is determined by the position of the mouse cursor.

• Moving Windows

• Another very configurable thing in X is how to move windows around.

• The most obvious method is to move the mouse cursor onto the *title bar* and drag the window around the screen. This may be done with any of the left, right, or middle buttons.

• Another way of moving windows may be holding down a key while dragging the mouse.

- Depth

- Since windows are allowed to overlap in X, there is a concept of *depth*.
- There are several operations that deal with depth:
  1. Raising the window
  2. Lowering the window
  3. Cycling through windows

- Iconization

- There are several other operations that can obscure windows or hide them completely.
- First is the idea of "iconization".
- Depending on the window manager, this can be done in many different ways.
- In *twm*, many people configure an icon manager.
- This is a special window containing a list of all the other windows on the screen

- Resizing

- There are several different methods to resize windows under X.
- It is dependent on your window manager and exactly how your window manager is configured.
- The method many Microsoft Windows users are familiar with is to click on and drag the border of a window.
- Another method used is to create a "resizing" button on the titlebar.
  - To resize windows, the mouse is moved onto the resize button and the left mouse button is held down.
  - You can then move the mouse outside the borders of the window to resize it.

- Maximization

- Most window managers support maximization.
- In twm, you can maximize the height, the width, or dimensions of a window. This is called "zooming" in *twm*'s language.

- Menus

- Another purpose for window managers is for them to provide menus for the user to quickly accomplish tasks that are done over and over.
- In general, different menus can be accessed by clicking on the root window, which is an immovable window behind all the other ones.

X Attributes

- There are many programs that take advantage of X.
- Some programs, like emacs , can be run either as a text-mode program or as a program that creates its own X window.
- However, most X programs can only be run under X.

• Geometry

- There are a few things common to all programs running under X.
- In X, the concept of geometry is where and how large a window's geometry has four components:
    1. The horizontal size. Usually measured in pixels.
    2. The vertical size, also usually measured in pixels.
    3. The horizontal distance from one of the sides of the screen.
    4. The vertical distance from either the top or the bottom.

- Display

- Every X application has a display that it is associated with.
- The display is the name of the screen that the X server controls.
- A display consists of three components:
  1. The machine name that the server is running on.
  2. The number of the server running on that machine.
  3. The screen number.

Common Features

- While X is a graphical user interface, it is a very uneven graphical user interface.
- It is impossible to say how any component of the system is going to work, because every component can easily be reconfigured, changed, and even replaced.
- Another cause of this uneven interface is the fact that X applications are built using things called "widget sets".
- Included with the standard X distribution are "Athena widgets".
- The other popular widget set is called "Motif".
- Motif is a commercial widget set similar to the user interface used in Microsoft Windows.

• Buttons

- Buttons are generaly the easiest thing to use.
- A button is invoked by positioning the mouse cursor over it and clicking the left button.
- Athena and Motif buttons are functionally the same.

• Menu Bars

• A menu bar is a collection of commands accessible using the mouse.

• Each word is a category heading of commands.

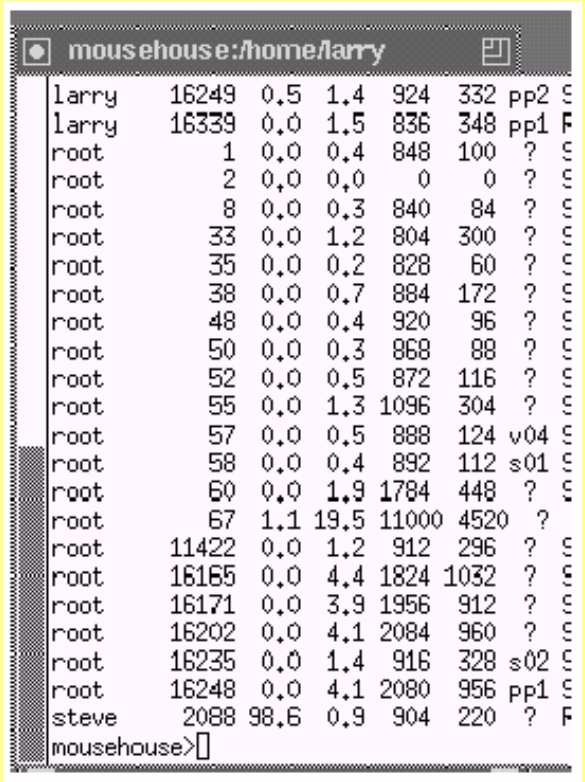
• File deals with commands that bring up new files and save files.

• To access a command, move the mouse cursor over a particular category and press and hold down the left mouse button.

• Scroll Bars

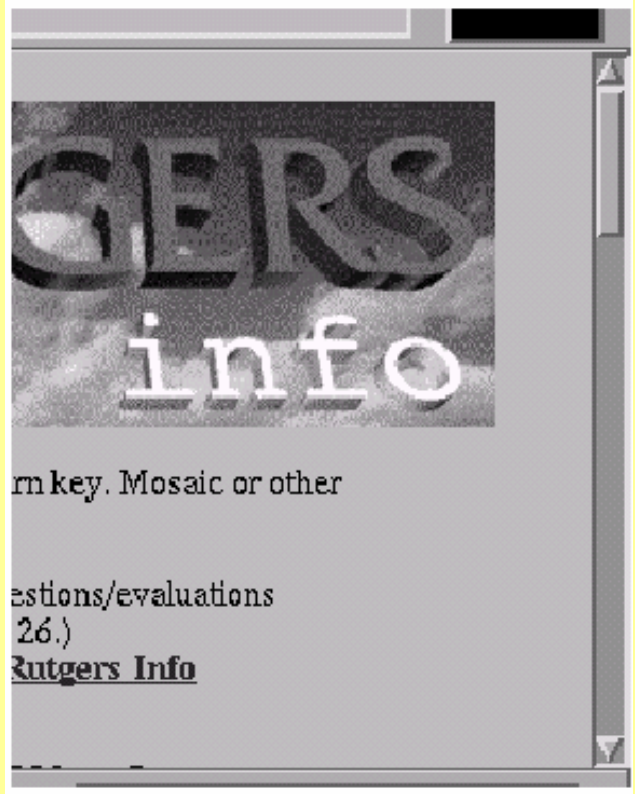
- A scroll bar is a method to allow people to display only part of a document, while the rest is off the screen.
- A vertical scroll bar may be to the left or right of the next and a horizontal one may be above or below, depending the application.

• Athena scroll bars operate differently from scroll bars in other windowing systems. Each of the three buttons of the mouse operate differently.



Motif Scroll Bars

- A motif scroll bar acts much more like a Microsoft Windows or Macintosh scroll bar.
- The behavior of clicking inside the scroll bar is widely different for Motif scroll bars than Athena scroll bars.
- The right button has no effect.
- Clicking the left button above the current position scroll upward.
- Clicking below the current position scroll downward.



- **history**

  - shell build a table of commands as you enter them and assign them a command number.

  - Each time you log in, the first command you enter is command 1, and the command number is incremented for each subsequent command that you enter.

  - You can review or repeat any previous command easily with just a few keystrokes.

  - To review your history in the shell, enter history at the prompt.

• There are three main mechanisms for working with the history list. You can specify a previous command

- by its command number,
- by the first word of the command, or,
- if you're working with the most recently executed command, by a special notation that easily fixes any mistakes you might have made as you typed it.

• If the 33rd command you entered was the who command, for example, you can execute it by referring to its command number: enter **!33** at the command line.

• You can execute it also by entering one or more characters of the command: **!w**, **!wh**, or **!who**

• You must enter enough characters to uniquely identify it in the history list.

C shell history commands

Command	Function
!!	Repeat the previous command
!$	Repeat the last word of the previous command
!*	Repeat all but the first word of the previous command
^a^b	Replace a with b in the previous command
!n	Repeat command n from the history list.

- **alias**

  - Using aliases, you easily can define new commands that do whatever you'd like, or even redefine existing commands to work differently, have different default flags, or more!

  - One of the most helpful aliases you can create specifies certain flags to ls so that each time you enter ls, the output will look as though you used the flags with the command.

  - to have the -l flags set: alias ls 'ls -l'

- If you’re coming from the DOS world, you might have found some of the UNIX file commands confusing.
- In DOS, for example, you use DIR to list directories, COPY to copy them, and so on.
- With aliases, you can recreate all
- those commands, mapping them to specific UNIX equivalents:
- # alias DIR ‘ls -l’
- # alias COPY ‘cp -i’
- # alias DEL ‘rm -i’

```
[bagriy@hyperion bagriy]$ > DIR
total 2
drwx------ 2 bagriy 512 Nov 25 11:39 Archives/
drwx------ 2 bagriy 1024 Dec 8 02:43 Mail/
```